GTSuite Server

Technical Documentation

Node.js Backend Server with Metadata-Driven Architecture

GTsoftware di Giancarlo Thiella

Generated: November 20, 2025

Table of Contents

1. Executive Summary

GTSuite Server is a sophisticated Node.js-based backend system developed by GTsoftware di Giancarlo Thiella. It serves as the core server infrastructure for a metadata-driven application framework that powers dynamic Angular/Ionic client applications.

Key Features

• **Metadata-Driven Architecture:** Pages, forms, menus, and UI components are defined through metadata stored in MongoDB

• **Multi-Database Support:** Native integration with MongoDB, Oracle, Microsoft SQL Server, PostgreSQL, and SQLite

• **Advanced Authentication:** OAuth 2.0 support (Google, Microsoft), JWT tokens, and optional 2FA/TOTP

• **Real-time Communication:** Socket.io integration for live updates and notifications

• **AI Integration:** Built-in support for OpenAI GPT-4, Anthropic Claude, and Google Gemini AI models

• **Document Processing:** Comprehensive support for DOCX, PDF, Excel, and XML document generation and manipulation

• **SQLite Import Pipeline:** Automated metadata import from Delphi-generated SQLite databases to MongoDB

• **Enterprise Features:** Scheduled tasks, email services, comprehensive logging, and backup capabilities

2. Architecture Overview

System Architecture

GTSuite follows a three-tier architecture with a metadata-driven approach that enables dynamic application behavior without code changes. The system is designed to support multiple client applications through a unified backend infrastructure.

Architecture Layers

Layer	Component	Description
Presentation	Angular/Ionic Clients	Dynamic UI rendered from metadata
Application	Express.js Server	RESTful API, business logic, authentication
Data	MongoDB Primary	Metadata storage (auth, data, logs)
	External Databases	Oracle, MSSQL, PostgreSQL integration
Integration	SQLite Import	Delphi application metadata pipeline

Data Flow

1. **Metadata Creation:** Delphi desktop application creates application definitions
2. **SQLite Export:** Metadata exported to SQLite database files
3. **Import Pipeline:** Node.js server imports SQLite data into MongoDB
4. **Client Request:** Angular/Ionic client requests page/menu metadata
5. **Metadata Delivery:** Server retrieves and translates metadata
6. **Dynamic Rendering:** Client dynamically renders UI based on metadata
7. **Data Operations:** Client executes CRUD operations through API
8. **External DB Access:** Server proxies requests to Oracle/MSSQL/PostgreSQL

3. Technology Stack

Core Technologies

Technology	Version/Package	Purpose
Runtime	Node.js	JavaScript runtime environment
Framework	Express.js 4.18.2	Web application framework
Primary Database	MongoDB 6.20.0	Metadata and application data storage
ODM	Mongoose 8.19.2	MongoDB object modeling
Authentication	JWT + Passport	Token-based auth with OAuth2
Real-time	Socket.io 4.7.1	WebSocket communication
Task Scheduling	node-cron 3.0.2	Scheduled job execution

Database Connectors

Database	Driver Package	Use Case
MongoDB	mongoose	Primary metadata storage
Oracle	oracledb 6.0.3	Enterprise database connectivity
SQL Server	mssql 11.0.1	Microsoft SQL Server integration
PostgreSQL	pg 8.11.3	PostgreSQL database access
SQLite	sqlite3 5.1.6	Metadata import from Delphi

AI & Document Processing

- **OpenAI Integration:** GPT-4o model support via openai 4.85.1
- **Anthropic Claude:** Claude Sonnet 4 integration via @anthropic-ai/sdk 0.36.3
- **Google Gemini:** Gemini 2.0 Pro via @google/generative-ai 0.22.0
- **Document Generation:** docxtemplater 3.60.1, @spfxappdev/docxmerger
- **PDF Processing:** @pspdfkit/nodejs for PDF manipulation
- **Excel Operations:** node-xlsx 0.24.0 for spreadsheet handling
- **XML Processing:** xml2js 0.6.2 for XML parsing and generation

4. Metadata System

The metadata system is the core architectural component of GTSuite. It enables the creation of dynamic, data-driven applications where the entire user interface, navigation structure, business rules, and data operations are defined through metadata rather than hardcoded in the application.

Metadata Pipeline

Stage 1: Design (Delphi Application)
The metadata design phase uses a Delphi desktop application that provides a visual interface for defining application structure, pages, forms, grids, actions, and business rules.

Stage 2: Export (SQLite Database)
Designed metadata is exported to SQLite database files, which serve as an intermediate portable format for metadata transfer.

Stage 3: Import (Node.js Server)
The GTSuite server provides API endpoints to import SQLite metadata files and transform them into MongoDB documents with proper structuring and optimization.

Stage 4: Consumption (Angular/Ionic Client)
Client applications request metadata through REST APIs and dynamically render the user interface based on the received metadata definitions.

Metadata Components

Component	Collection	Purpose
Pages	GtsPages	Page definitions with all UI components
Menus	GtsMenu	Hierarchical navigation structure
Forms	Embedded in Pages	Form layouts and field definitions
Grids	Embedded in Pages	Data grid configurations
Actions	Embedded in Pages	User actions and event handlers
Views	Embedded in Pages	Data view definitions
Toolbars	Embedded in Pages	Toolbar button configurations
Data Adapters	Embedded in Pages	Data source connections
SQL Specs	GtsSql	SQL query templates
Reports	GtsReports	Report definitions and templates
Messages	Embedded in Pages	User-facing messages
Conditional Rules	Embedded in Pages	Business logic rules

Page Metadata Structure

```
{
  "prjId": "GTSW",
  "formId": 1001,
  "formUrl": "/dashboard",
  "pageTitle": "Dashboard",
  "initAction": "loadData",
  "tabs": [...],
  "toolbars": [...],
  "views": [...],
  "actions": [...],
  "dataSets": [...],
  "forms": [...],
  "grids": [...],
  "pageFields": [...],
  "condRules": [...],
  "messages": [...],
  "reportsGroups": [...],
  "sqls": [...]
}
```

5. Database Architecture

MongoDB Database Structure

GTSuite uses a multi-database MongoDB architecture with three separate databases for different data domains. This separation provides better organization, security, and performance optimization.

Database	Purpose	Key Collections
GTSauth	Authentication & Authorization	GtsProject, GtsUser, GtsAuthProfiles, GtsRoles, GtsObjectRoles, GtsAuthKeys, GtsPasswordPolicies, GtsAuthParams
GTSdata	Application Metadata	GtsPages, GtsMenu, GtsSql, GtsReports, GtsMailMerge, GtsMLText, GtsInstructionsAI, Sequences, GtsDataSchemas
GTSlogs	System Logs	GtsAuthLog, GtsErrorLog, GtsDBLog, GtsSqliteLog, GtsReportsLog, GtsScheduledTaskLog, GtsMails

Key Collections Detail

GtsProject: Project definitions with configuration and database connections
GtsUser: User accounts with authentication credentials and profile settings
GtsPages: Complete page metadata including all embedded UI components
GtsMenu: Hierarchical menu structure with tree-based organization
GtsAuthProfiles: Role-based access control profiles
GtsRoles: User roles and permissions
GtsObjectRoles: Object-level permission mappings
GtsDBConnections: External database connection configurations
GtsScheduledTasks: Cron job definitions and schedules
GtsMailService: Email service configuration
GtsReports: Report templates and definitions
GtsSql: SQL query specifications and templates

External Database Integration

GTSuite supports seamless integration with external enterprise databases. Database connections are configured per project and stored in the GtsDBConnections collection. The server provides a unified API for executing queries and procedures across different database platforms while handling platform-specific syntax and connection management.

- **Oracle:** Full support for PL/SQL procedures, packages, and Oracle-specific features
- **Microsoft SQL Server:** T-SQL procedures, views, and SQL Server integration
- **PostgreSQL:** PostgreSQL functions, procedures, and advanced query features

- **SQLite:** Lightweight database support primarily for metadata import

6. API Documentation

GTSuite exposes a comprehensive REST API organized into logical route groups. All API endpoints require authentication via JWT tokens, and most support role-based access control.

API Route Groups

Route Prefix	Module	Purpose
/api/auth	gts-auth	Authentication and authorization
/api/user	gts-user	User management and profiles
/api/prj	gts-projects	Project configuration
/api/setup	gts-setup	System setup and configuration
/api/data	gts-data	Metadata retrieval and management
/api/sqlite	gts-sqlite	SQLite metadata import
/api/db	gts-db	External database operations
/api/files	gts-file	File upload and management
/api/mail	gts-mail	Email sending services
/api/task	gts-tasks	Scheduled task management

Key API Endpoints

Authentication Endpoints (/api/auth)

- POST /api/auth/login - User login with credentials
- POST /api/auth/register - New user registration
- POST /api/auth/checkAuth - Validate JWT token
- POST /api/auth/verify2fa - Two-factor authentication verification
- GET /auth/google - Google OAuth login
- GET /auth/ms - Microsoft OAuth login

Metadata Endpoints (/api/data)

- POST /api/data/getPageData2 - Retrieve page metadata with translations
- POST /api/data/getMenuData - Get menu structure with translations
- POST /api/data/getReportData - Execute report and retrieve data
- POST /api/data/getSQLSpec - Get SQL specification for data operations

- POST /api/data/getStdMLText - Retrieve standard multilanguage text
- POST /api/data/saveMLText - Save multilanguage translations
- POST /api/data/createSequence - Create new sequence generator
- POST /api/data/getNewSequence - Get next sequence number

SQLite Import Endpoints (/api/sqlite)

- POST /api/sqlite/loadSetup - Import setup configuration from SQLite
- POST /api/sqlite/loadPage2 - Import complete page metadata from SQLite
- POST /api/sqlite/openConn - Open SQLite database connection
- POST /api/sqlite/closeConn - Close SQLite database connection
- POST /api/sqlite/removePageData - Remove existing page data before import

Database Endpoints (/api/db)

- POST /api/db/execProc - Execute stored procedure on external database
- POST /api/db/execQuery - Execute SQL query on external database
- POST /api/db/testConnection - Test database connection
- POST /api/db/getSchema - Retrieve database schema information

7. Authentication & Security

Authentication Mechanisms

GTSuite implements a multi-layered authentication system supporting various authentication methods to meet different security requirements and user preferences.

- **JWT Token Authentication:** Primary authentication using JSON Web Tokens with configurable expiration
- **OAuth 2.0 Integration:** Support for Google and Microsoft OAuth authentication flows
- **Two-Factor Authentication (2FA):** Optional TOTP-based 2FA using otplib
- **Session Management:** Express sessions with configurable session store
- **Password Policies:** Configurable password complexity and expiration rules
- **Bcrypt Password Hashing:** Industry-standard password hashing with bcrypt

JWT Token Structure

```
{
  "userId": "user123",
  "email": "user@example.com",
  "prjId": "GTSW",
  "roles": ["admin", "user"],
  "iat": 1634567890,
  "exp": 1634654290
}
```

Role-Based Access Control (RBAC)

GTSuite implements a sophisticated RBAC system with multiple levels of permission control:

- **Project Level:** Users are assigned to projects with specific roles
- **Role Level:** Roles define sets of permissions (GtsRoles collection)
- **Profile Level:** Authentication profiles define access patterns (GtsAuthProfiles)
- **Object Level:** Fine-grained permissions on specific objects (GtsObjectRoles)
- **Menu Level:** Menu items can be filtered based on user roles

Security Features

- **CORS Protection:** Configurable cross-origin resource sharing with whitelist
- **Rate Limiting:** Protection against brute force attacks
- **Encryption:** AES-256-CTR encryption for sensitive data (cryptoService)
- **Input Validation:** Request validation and sanitization
- **SQL Injection Prevention:** Parameterized queries and ORM usage
- **XSS Protection:** Content security headers and input escaping
- **HTTPS Support:** TLS/SSL configuration for secure communication
- **Security Logging:** Comprehensive authentication and authorization logging

8. Configuration

GTSuite is configured through environment variables defined in the .env file. This approach provides flexibility for different deployment environments (development, test, production) without code changes.

Server Configuration

Variable	Description	Example
PORT	Server listening port	3000
HTTPS	Enable HTTPS (Y/N)	N
NODEJS_SERVER	Base server URL	http://localhost
CORS	Allowed origin for CORS	http://localhost:4200
ACTIVATE_CRON	Enable scheduled tasks	Y
ACTIVATE_LOG	Enable logging	Y

Database Configuration

Variable	Description	Example
MONGO_SERVER	MongoDB server address	cluster0.mongodb.net
MONGO_USER	MongoDB username	GTSuite
MONGO_PW	MongoDB password	********
MONGO_AUTH_DB	Authentication database	GTSauth
MONGO_DATA_DB	Metadata database	GTSdata
MONGO_LOGS_DB	Logs database	GTSlogs

Authentication Configuration

Variable	Description
JWT_SECRET	Secret key for JWT token signing
JWT_UNIQUE_VALIDATION	Enable unique token validation (Y/N)
SIGN_WITH_GOOGLE	Enable Google OAuth (Y/N)
GOOGLE_CLIENT_ID	Google OAuth client ID
GOOGLE_CLIENT_SECRET	Google OAuth client secret
SIGN_WITH_MICROSOFT	Enable Microsoft OAuth (Y/N)
MICROSOFT_CLIENT_ID	Microsoft OAuth client ID

TOTP_ENABLED	Enable two-factor authentication (Y/N)
TOTP_ISSUER	TOTP issuer name

File System Configuration

- DIRECTORY_FILE_BASE - Base directory for uploaded files
- DIRECTORY_SQLITEDATA_BASE - Directory for SQLite import files
- DIRECTORY_REPORTS_BASE - Directory for generated reports
- BACKUP_PATH - Directory for database backups
- MONGO_TOOLS_PATH - Path to MongoDB tools (mongodump, mongorestore)

9. Installation & Deployment

Prerequisites

- **Node.js:** Version 18.x or higher recommended
- **MongoDB:** Version 6.0 or higher (local or MongoDB Atlas)
- **Oracle Instant Client:** Required if using Oracle database connectivity
- **npm:** Node package manager (included with Node.js)
- **Git:** For source code management

Installation Steps

Clone and navigate to project:

```
cd /path/to/project
```

Install dependencies:

```
npm install
```

Configure environment:

```
cp .env.example .env
# Edit .env with your configuration
```

Create directories:

```
mkdir -p Files SQLLiteData reports Backup
```

Start server:

```
npm start
# Or for development: npm run dev
```

Production Deployment

For production deployment, consider the following best practices:

- **Process Manager:** Use PM2 or similar to manage the Node.js process

- **Reverse Proxy:** Configure Nginx or Apache as reverse proxy
- **HTTPS:** Enable HTTPS with valid SSL certificates
- **Environment:** Set SETUP_DB_MODE=P for production
- **Logging:** Configure proper log rotation and monitoring
- **Backups:** Implement automated database backup strategy
- **Security:** Use firewall rules and restrict database access
- **Monitoring:** Set up application and server monitoring

PM2 Configuration

Install PM2 globally:

```
npm install pm2 -g
```

Start application:

```
pm2 start server.mjs --name gts-server
```

Auto-restart on reboot:

```
pm2 startup
pm2 save
```

Monitor application:

```
pm2 monit
```

View logs:

```
pm2 logs gts-server
```

Nginx Reverse Proxy Configuration

Example Nginx configuration for reverse proxy:

```
server {
    listen 80;
    server_name your-domain.com;
    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
    }
}
```

10. Development Guide

Project Structure

The GTSuite server follows a modular architecture with clear separation of concerns:

```
/GTS
■■■ app.mjs                    # Express application
■■■ server.mjs                 # HTTP/HTTPS server
■■■ package.json               # Dependencies
■■■ .env                       # Configuration
■■■ controllers/               # Request handlers
■■■ routes/                    # API routes
■■■ models/                    # Mongoose schemas
■   ■■■ components/            # Metadata models
■   ■■■ logs/                  # Log models
■■■ services/                  # Business services
■■■ middleware/                # Express middleware
■■■ scripts/                   # Utility scripts
```

Adding New API Endpoints

To add new API endpoints, follow this pattern:

```
// 1. Create controller function
export class MyController {
    static async myFunction(req, res, next) {
        try {
            const { param1, param2 } = req.body;
            const result = await operation(param1, param2);
            res.status(200).json({
                success: true,
                data: result
            });
        } catch (error) {
            res.status(500).json({
                success: false,
                error: error.message
            });
        }
    }
}

// 2. Add route
router.post("/myEndpoint",
    GtsAuthCtrl.checkAuth,
    MyController.myFunction
);

// 3. Register in app.mjs
app.use("/api/my", myRouter);
```

Creating Mongoose Models

```
import { Schema } from "mongoose";
import { connGTSdata } from "../services/connections.mjs";

const mySchema = new Schema({
    prjId: { type: String, required: true },
    name: { type: String, required: true },
    description: { type: String },
    data: { type: Object },
    items: { type: Array, default: [] },
    createdAt: { type: Date, default: Date.now }
});

mySchema.index({ prjId: 1, name: 1 });

export const MyModel = connGTSdata.model('MyModel', mySchema);
```

Database Service Pattern

For database operations, use service classes to encapsulate database-specific logic:

```
export class DatabaseService {
    static async executeQuery(connection, query, params) {
        try {
            const result = await connection.query(query, params);
            return { success: true, data: result };
        } catch (error) {
            return { success: false, error: error.message };
        }
    }

    static async executeProcedure(connection, procName, params) {
        // Stored procedure execution
    }
}
```

Error Handling Best Practices

- **Always use try-catch blocks** in async functions
- **Return consistent error responses** with success flag and error message
- **Log errors** to the appropriate log collection (GtsErrorLog)
- **Use appropriate HTTP status codes** (200, 400, 401, 403, 500)
- **Never expose sensitive information** in error messages
- **Validate input parameters** before processing
- **Handle database connection errors** gracefully

11. Project Structure

Controllers

Controller	Size	Purpose
gts-data.mjs	84KB	Metadata retrieval and management
gts-sqlite2.mjs	103KB	SQLite metadata import
gts-user.mjs	75KB	User management and authentication
gts-db.mjs	36KB	External database operations
gts-setup.mjs	28KB	System configuration
gts-projects.mjs	13KB	Project management
gts-scheduledTasks.mjs	14KB	Cron job management
gts-mail.mjs	11KB	Email operations
gts-ai.mjs	3.5KB	AI service integration

Services

Service	Purpose
mongoDBService.mjs	MongoDB operations and connection management
oracleService.mjs	Oracle database connectivity and query execution
mssqlService.mjs	Microsoft SQL Server integration
postgresService.mjs	PostgreSQL database operations
sqliteService.mjs	SQLite database handling for imports
allDatabaseService.mjs	Unified database interface
authService.mjs	Authentication and token management
cronService.mjs	Scheduled task execution
logsService.mjs	Logging operations
cryptoService.mjs	Encryption and decryption
files.mjs	File upload and management
openaiService.mjs	OpenAI API integration
claudeaiService.mjs	Anthropic Claude API integration
geminiaiService.mjs	Google Gemini API integration

Models Organization

Models are organized into logical groups:

- **Root Models:** User, Project, Authentication, Configuration
- **components/:** Page metadata components (pages, menus, reports, SQL, forms, etc.)
- **logs/:** System logging models (auth logs, error logs, database logs, etc.)

12. Integration Points

Client-Server Integration

The GTSuite server is designed to work seamlessly with Angular and Ionic client applications. The metadata-driven architecture allows clients to dynamically render user interfaces without requiring client-side code changes when modifying application structure.

1. **Authentication:** Client obtains JWT token via /api/auth/login
2. **Project Selection:** Client retrieves available projects
3. **Menu Loading:** Client requests menu structure via /api/data/getMenuData
4. **Page Rendering:** Client requests page metadata via /api/data/getPageData2
5. **Data Operations:** Client executes CRUD operations through API
6. **Real-time Updates:** Socket.io connection for live notifications
7. **File Operations:** Upload/download through /api/files endpoints
8. **Report Generation:** Report execution through /api/data/getReportData

Delphi Application Integration

The Delphi desktop application serves as the metadata design tool. It creates SQLite databases containing complete application definitions that are then imported into the GTSuite server.

Design Phase: Use Delphi application to design pages, forms, and business logic

Export Phase: Export metadata to SQLite database file

Transfer Phase: Upload SQLite file to server's DIRECTORY_SQLITEDATA_BASE

Import Phase: Execute /api/sqlite/loadPage2 or /api/sqlite/loadSetup

Verification Phase: Check import logs in GtsSqliteLog collection

Deployment Phase: Metadata immediately available to client applications

External System Integration

- **Email Services:** SMTP integration via nodemailer for transactional emails
- **OAuth Providers:** Google and Microsoft OAuth 2.0 authentication
- **AI Services:** OpenAI, Anthropic Claude, and Google Gemini API integration

- **Document Services:** LibreOffice integration for document conversion
- **Enterprise Databases:** Oracle, SQL Server, PostgreSQL connectivity
- **Backup Services:** MongoDB backup utilities integration
- **ReCAPTCHA:** Google reCAPTCHA for bot prevention

Conclusion

GTSuite Server represents a sophisticated, enterprise-grade backend system that enables rapid application development through its metadata-driven architecture. By separating application structure from code, it allows for unprecedented flexibility and maintainability. The system's multi-database support, comprehensive authentication mechanisms, and extensive integration capabilities make it suitable for a wide range of enterprise applications, from simple CRUD operations to complex business processes. For additional support or inquiries, please contact GTsoftware di Giancarlo Thiella.